Extended Abstract- Using Rescheduling and Flexible Execution to Address Uncertainty in Execution Duration for a Planetary Rover

Jagriti Agrawal, Wayne Chi, Steve Chien
Jet Propulsion Laboratory, California Institute of Technology
{firstname.lastname}@jpl.nasa.gov

Abstract

The paper "Using Rescheduling and Flexible Execution to Address Uncertainty in Execution Duration for a Planetary Rover" [Agrawal et al. 2019] discusses several rescheduling and execution techniques to allow a scheduler to respond effectively to changes in execution, such as activities ending earlier or later than expected. We discuss these techniques both theoretically and practically in the context of limited CPU, nonzero runtime, embedded scheduler intended for NASA's next planetary rover.

1 Introduction

The paper summarized by this extended abstract [Agrawal et al., 2019] subsumes the paper titled "Embedding a Scheduler in Execution for a Planetary Rover" [Chi et al. , 2018] which was accepted to ICAPS 2018. The previously accepted paper [Chi et al., 2018] discusses the rescheduling methods and includes some discussion of Flexible Execution. However, it does not include discussion of activities taking longer than expected.

In the current paper, in addition to the rescheduling methods, we present two algorithms to address how to handle activities running long. Furthermore, the empirical results we present in the current paper are more extensive both because more realistic inputs have become available since the submission of the previous paper and the current results include analysis based on additional factors such as the scheduler taking less time to run than expected and activities taking longer than expected. We also include significantly more detailed empirical analysis on how well our defined computation model to measure performance applies to our problem inputs, as well as which factors contribute to the inaccuracy of the theoretical model as it applies to our inputs.

2 Goal of the Scheduler

The goal of the scheduler [Rabideau and Benowitz, 2017] is primarily to schedule all activities and secondly to schedule activities such that the schedule has the shortest possible *makespan*, where the makespan is the difference between the latest time an activity is scheduled to end and the earliest time an activity is scheduled to start. A shorter makespan implies more resources such as time, energy, and data volume are freed.

3 Non-zero Scheduler Runtime

Since the scheduler is invoked multiple times during execution to generate updated schedules and has a nonzero runtime, we must know what to execute while the scheduler is running. We choose to use the concept of a *commit window* to execute activities based on the previously generated schedule. The commit window is in interval of fixed duration from the current time and any activity whose scheduled start time is within the commit window is committed and cannot be rescheduled by the scheduler. We use a commit window that is equal to the predicted scheduler run time, T_{sc_p}. So, even if the scheduler takes less time than expected to run, it cannot reschedule to start activities earlier than now + T_{sc_p}. We apply a high margin so we assume the scheduler will not take more time than expected to run.

4 Framework for Analysis

The framework we use to quantify the performance of the rescheduling and Flexible Execution methods quantifies how the techniques are able to reduce the makespan by mitigating two kinds of losses which result in unusable time and a longer makespan:

- *Scheduler runtime loss*- the time that the scheduler is unable to recoup while the scheduler is running
- *Scheduler invocation loss*- the time lost due to waiting to reinvoke the scheduler (e.g. not invoking the scheduler immediately after an activity ends early)

5 Rescheduling Methods

We discuss two rescheduling methods. Using *Fixed Cadence Scheduling*, the scheduler is invoked periodically every *C* seconds. Since the scheduler is invoked at a predetermined time using this method, it is often unable to react effectively to changes in execution, resulting in both scheduler runtime loss and scheduler invocation loss. Using *Event Driven Scheduling,* the scheduler is invoked in response to changes in execution (e.g. if an activity ends later or earlier than expected by some threshold or if it is vetoed and removed from the current schedule to be considered in the next scheduler invocation). Since the scheduler is invoked immediately after an activity ends early, scheduler invocation loss is eliminated if the threshold for Event Driven Scheduling is set accordingly.

6 Flexible Execution

We discuss Flexible Execution (FE) as a method to update start times of activities when the scheduler is not running. We discuss two FE algorithms, *Extended Veto* and *Extended Push*. Both algorithms perform the same when activities end earlier than expected but they vary when activities run late. We use the concept of a *dispatch window*, a fixed amount of time after Now. FE is only able to modify start times of activities which are scheduled to start within the dispatch window. We also use the predecessor-successor relationships between activities, which affect whether activity start times can be adjusted. The predecessor-successor relationship (e.g. Activity A must complete successfully before Activity B starts) is one of relative ordering between activities that share the same unit resources or establish precedence between one another.

6.1 Extended Push

We discuss how FE handles activities running late. Using the Extended Push algorithm, if a predecessor activity runs long, then the start time of the successor activity is pushed until the predecessor activity completes or until an execution time constraint (dictating the time windows during which the activity is allowed to occur) is violated. If multiple consecutive activities must be pushed because they are in a chain of predecessor-successor relationships, then the chained activities will continue to be pushed as long as execution constraints are not violated.

6.2 Extended Veto

Using the Extended Veto algorithm, the successor activity will be pushed up to some amount, *M,* but it may also be vetoed under certain conditions.

7 Empirical Evaluation

In order to evaluate each of the strategies, we apply them to various sets of inputs called *sol types* comprised of activities and their constraints. *Sol types* are currently the best available data on expected Mars 2020 rover operations. We use the *M2020 surrogate scheduler* to construct a schedule from the inputs and simulate execution. The surrogate scheduler is the same implementation as the M2020 onboard scheduler but runs much faster as it is intended for a Linux workstation. We use 8 different sol types where each sol type contains between 20 and 40 activities. Since data from MSL indicates that activities completed on average 28 percent early, for our model to determine activity execution durations, we choose from a normal distribution where the mean is 72 percent of the original, predicted duration ([Gaines et al., 2016], [JPL, 2017]). The standard deviation determines the percentage of activities that will take longer to execute than expected.

7.1 Conclusions from Analysis

From our analysis we conclude that Event Driven scheduling with Extended Push FE algorithm results in the highest percentage of activities successfully executed when the number of activities that longer than expected is varied. When activities never take longer than expected, Event Driven scheduling with either FE algorithm results in the highest makespan gain.

Using our computational model for scheduler runtime and invocation loss, we find that Fixed Cadence Scheduling results in the lowest makespan gain and the most loss. Event Driven scheduling decreases both scheduler runtime and scheduler invocation loss compared to Fixed Cadence scheduling and FE with Event Driven scheduling further reduces scheduler invocation loss. We also conclude that execution time constraints (earliest and latest times activities are allowed to start), setup activities (required activities such as preheats that must occur before the activity), and parallel activities (activities not in the critical path) contribute to the inaccuracy of our computational model for scheduler runtime and invocation loss. Removing these factors from the sol types result in an accurate model.

Acknowledgments

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[Agrawal et al., 2019] Agrawal, J.; Chi, W.; Chien, S. 2019. Using Rescheduling and Flexible Execution to Address Uncertainty in Execution Duration for a Planetary Rover. In Review.

[Chi et al., 2018] Chi, Wayne, et al. "Embedding a scheduler in execution for a planetary rover." *Twenty-Eighth International Conference on Automated Planning and Scheduling*. 2018.

[Gaines et al., 2016] Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; et al. 2016. Productivity challenges for mars rover operations: A case study of mars science laboratory operations. Technical report, Technical Report D-97908, Jet Propulsion Laboratory.

[JPL, 2017] Jet Propulsion Laboratory. 2017. Mars science laboratory mission https://mars.nasa.gov/msl/2017-11-13.

[Rabideau and Benowitz, 2017] Rabideau, Gregg, and Ed Benowitz. "Prototyping an onboard scheduler for the mars 2020 rover." *Proceeding of International Workshop on Planning and Scheduling for Space, Pittsburgh, PA*. 2017.